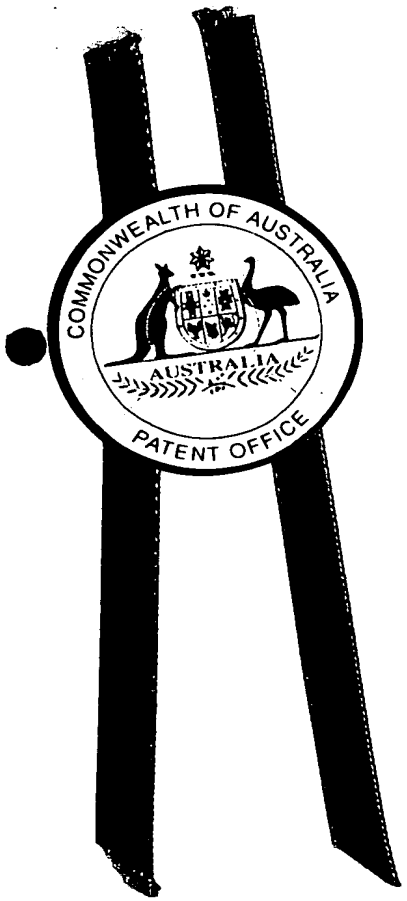
PCT/AU00/00651

AU 00/651
4

Patent Office
Canberra

I, ANNA MAIJA EVERETT, ACTING TEAM LEADER EXAMINATION SUPPORT & SALES hereby certify that annexed is a true copy of the Provisional specification in connection with Application No. PQ 4668 for a patent by TELSTRA R&D MANAGEMENT PTY. LTD. filed on 15 December1999.

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)



WITNESS my hand this
Twenty-seventh day of June 2000

A.M. Everett

ANNA MAIJA EVERETT
ACTING TEAM LEADER
EXAMINATION SUPPORT & SALES

P/00/009 28/5/91
Regulation 3.2

TELSTRA R&D MANAGEMENT PTY. LTD.

AUSTRALIA

Patents Act 1990

PROVISIONAL SPECIFICATION

for the invention entitled:

"A METHOD OF DEVELOPING AN INTERACTIVE SYSTEM"

The invention is described in the following statement:

A METHOD OF DEVELOPING AN INTERACTIVE SYSTEM

The present invention relates to a method of developing an interactive system and, in particular to a development system and tools for generating an application for an interactive system.

Interactive systems, such as interactive voice response systems (IVRs), are able to communicate with other machines or humans using natural language dialogue. The systems are able to prompt a communicating party for data required to execute application tasks and need to cater for a wide variety of possible responses to the prompts, particularly when communicating with humans. Developing a set of rules which defines all of the possible answers or responses to the prompts is particularly problematic and labour intensive. Also developing a structure to manage the dialogue which occurs between the communicating parties, is complex. Accordingly, it is desired to provide a method and tools which facilitates application development or at least provides a useful alternative.

In accordance with the present invention there is provided a method of developing an interactive system, including:

inputting application data representative of an application for said system, said application data including operations and parameters for said application;

generating prompts on the basis of said application data; and

generating grammar on the basis of said application data.

Advantageously, the method may also include generating a dialogue state machine on the basis of said application data. The grammar and state machine preferably include slots defining data on which the system executes the operations.

Advantageously, the method may include executing grammatical inference to extend the grammar. This may involve a model merging process including processing rules of the grammar, creating additional rules representative of repeated phrases, and merging equivalent symbols of the grammar.

The present invention also provides a system for developing an interactive system, including:

means for inputting application data representative of an application for said system, said application data including operations and parameters for said application;

means for generating prompts on the basis of said application data; and

means for generating grammar on the basis of said application data.

The present invention also provides development tools for an interactive system, including:

code for inputting application data representative of an application for said system, said application data including operations and parameters for said application;

code for generating prompts on the basis of said application data; and

code for generating grammar on the basis of said application data.

Preferably the application data is inputted in an application file, and the operations include a number of input and return parameters with parameter types.

The present invention also provides a grammatical inference method for developing grammar, including processing rules of the grammar, creating additional rules representative of repeated phrases, and merging equivalent symbols of the grammar.

Preferred embodiments of the present invention are hereinafter described, by way of example only, with reference to the accompanying drawings, wherein:

Figure 1 is a block diagram of a preferred embodiment of an interactive system connected to a communications network;

Figure 2 is a more detailed block diagram of the interactive system;

Figure 3 is a preferred embodiment of a development system for the interactive system;

Figure 4 is a flow diagram of a model merging process of the development system;

Figure 5 is a flow of a phrase addition phase of the model merging process;

Figure 6 is a flow diagram of a merging phase of the model merging process;

Figure 7 is a flow diagram of rule uniqueness procedure of the model merging process;

Figure 8 is a schematic diagram illustrating appending a grammar symbol;

Figure 9 is a schematic diagram illustrating reusing a rule of a grammar symbol;

Figure 10 is schematic diagram illustrating creating a grammar rule;

Figure 11 is a schematic diagram illustrating deleting a grammar rule;

Figure 12 is a schematic diagram of merging grammar symbols; and

Figure 13 is a schematic diagram of iterations in the merging process.

An interactive system, as shown in Figure 1, is able to communicate with another party, being a human or machine, using a natural language dialogue. A communication path is established with the party over a communications network 4, such as the PSTN and/or Internet. The path is between the system 2 and a communications terminal, such as a standard telephone 6 or computer system 8. When the communicating parties are human and a voice terminal, such as the telephone 6, is used the system 2 converts prompts for the party to speech for delivery to the terminal, and interprets speech received from the terminal. When communicating with a machine, such as the computer 8, text data representative of the prompts and responses can be passed between the machines 2 and 8. The architecture of the system 2 can be considered to be divided into three distinct layers 10 to 14, an application layer 10, a natural language processing layer 12 and a signal processing layer 14. The application layer 10 defines an application for the system 2, such as a bill payment service, a customer service operation or a sales service. The application identifies the operations and transactions to be executed by the system 2. The natural language processing layer 12 defines the prompts to be generated by the system 2, the grammar which is acceptable in return and the different operation states and state transitions of the system 2. The signal processing layer 14 handles the communications interface with the network 4, the terminals 6 and 8, and the data processing required for the interface, such as speech generation and recognition.

The prompt layer 12, as shown in Figure 2, includes a finite state machine (FSM) 20, the prompts 22 and the grammar 24. The FSM 20 controls the states for the system 2 and the state transition and accordingly controls the decisions and tasks executed by the system 2. The FSM 20 effectively performs dialogue management for the system 2, so as to control selective transmission of the prompts 22 and act in response to the grammar 24. The prompts 22 are words of questions which can be sent by the system 2. The grammar 24 is a set of rules stipulating words and/or phrases which form acceptable answers to the prompts. The rules also define slots in the answers which constitute parameters required by the FSM 20 for execution of decisions and tasks. Accordingly, the slots are also defined in the FSM 20. The prompts, on

instruction from the FSM 20, are selectively passed to a speech generator 26 for conversion into speech or voice and transmission to the terminals 6 or 8 using a communications interface 30 of the system 2. Responses received from a party are received by the communications interface 30 and are converted to representative data by a speech recognition module 28. The response data is passed by the module 28 to the grammar 24 for processing. The grammar 24 interprets the responses and passes operation instructions and operation data from the slots to the FSM 20.

The interactive system 2 may take a number of forms, as will be understood by those skilled in the art, ranging from a hand held mobile device to a distributed computer system. For simplicity, the preferred embodiments are described hereinafter with reference to the interactive IVR being an IVR. IVRs are produced by a number of telecommunications equipment providers, for example the Voice Processing Service (VPS) by Periphonics Inc. The parts 26, 28 and 30 of the signal processing layer are normally standard hardware and software components provided with the IVR. A developer would then be required to define at length program code, which can be compiled, for the components 20, 22 and 24 of the prompt layer 12. This is an extremely onerous task which the preferred embodiments seek to alleviate.

A development system 40, as shown in Figure 3, includes an application generator 42 which operates on an application file 44 to generate an FSM 20, prompts 22 and grammar 24. The grammar 24 is refined by a grammatical inference engine 46. The application file 44 can be considered to define the application layer 10. The file 44 includes semantic data which describes semantics to be used in the IVR 2. The file also defines the operations to be executed by the system 2 and the parameters required to execute the operations. An example of an application file 44 for a stock trading application is shown in Appendix 1. The operations defined are "buy", "sell" and "quote". The operations each have a number of input parameters defined by a parameter name and a number of output or return parameters defined by a return name, with all of the parameters being allocated a parameter type. The parameter types are predefined in the IVR and include integers, real numbers, dates, times and money amounts. Parameters types can also be defined by the developer in a list of items. The example of Appendix 1 includes a list of stocks to be used by the application. A list of items may be products, companies, options or any item of interest. The operations "sell" and "quote" are defined as requiring a user to be prompted to confirm that the input parameters are correct before proceeding with the operation. The application file can also be used to define additional

information that can be obtained from a user of the IVR, such as a user's account number and PIN. Initial data such as this can be considered to be preamble parameters which are collected before a top-level state of the FSM 20 is entered, and are used as global variables. Global variables for the IVR 2 are used in all operations of the IVR. The names of the operations are used in the grammar 24 as keywords to signify execution of the operation. For example, a user may respond to a prompt by saying "sell" or "sell all my holdings in BHP".

The finite state machine 20, the prompts 22 and the grammar 24 generated by the application generator 42 on the basis of the application file 44 of Appendix 1 are shown in Appendices 2, 3 and 4, respectively. A slot is defined for each input parameter name, as well as a slot for "operation". The slots are therefore operation, stockname, number and price.

The finite state machine 20 of the stock trading example is written in the ITUs Specification Description Language (SDL) of the ITU. The FSM 20 includes a number of procedures with variables, and are similar to subroutines in that a stack is used to control each procedure's behaviour, yet, a procedure can include states. An FSM 20 is generated by the application generator 42 executing the following steps:

(a) Create an initial top-level state.

(b) For each operation of the application file 44 create a state transition from the top-level state.

(c) For each parameter create a variable. The name of the variable is globally unique, which is achieved by pre-pending the operation name to the parameter name.

(d) For each state transition related to an operation, set all input parameters that may be obtained in a first response from a user to a value, otherwise reset the parameters to a value representing "null" or "unknown". A call is made to a nested state machine, such as an SDL procedure, to check each parameter in turn. If an operation needs to be confirmed, a procedure is established to ask the user for confirmation, otherwise the operation is executed. The results of the operation are passed to the output parameters and forwarded to the user prior to returning to the top-level state.

(e) A number of procedures are established for generic operations for IVRs, such as help, cancel, repeat and operator.

(f) For each input parameter that does not have a default, a nested procedure is established to determine if a parameter has been set to a value obtained by a response. If it is not set, a default is used.

(g) For each input parameter a nested procedure is created to check if an incoming message or response sets a parameter. If the parameter is set in the incoming message, the variable relating to the parameter is set.

(h) For each parameter that does not have a default, a nested procedure is created to see if the parameter has been already set. If it is not set, the procedure prompts the user for the parameter.

The grammar 24, as shown in Appendix 4, has a hierarchical structure, similar to a tree structure, where the nodes can be considered to be symbols that are either terminal or non-terminal. A terminal symbol represents a specific word. A non-terminal symbol represents other symbols and can be expanded into a set of symbols using a rule of the grammar. A rule defines a relationship between a non-terminal and symbols it can represent. For example, buy, sell, cancel, repeat, help and quit are terminals, whereas CommonStockCmds and WaitAskbuystockname are non-terminals. The grammar of Appendix 4 is a context free grammar written in the Nuance grammar format. The grammar is generated by executing the following steps:

(a) For each state in the FSM 20 a top-level grammar is created.

(b) For each top-level grammar a common set of commands are included, e.g. cancel, help, repeat and quit.

(c) A non-terminal is created for each enumerated item from the application file 44.

(d) For each operation a non-terminal is created that represents that different words can be used to request that operation, and the operation name is placed into the non-terminal.

(e) For the top-level grammar attached to the top-level state, e.g. TopLevelStock, a non-terminal is added for each operation, to which is added are attribute containing the operation slot. An attribute is a set of key value pairs defining a slot value. For example the attribute "{(operation buy)}" is added to the non-terminal "GR_buy".

(f) For each top-level grammar that corresponds to a state requesting an input parameter, a non-terminal is added that returns to the FSM an element of the

requested type. Predefined rules are established for non-terminals that represent predefined types of parameters, e.g. money and integer.

The prompts 22 of Appendix 3 are generated from the application file 44 and for the example are written in the Clips language. The application generator 42 generates the prompts by executing the following steps:

(a) The initial top-level prompt is set to be "Welcome to the stock application. Please say one of the following". The prompt then lists the names of the operations, e.g. "buy, sell or quote".

(b) For the prompts where a parameter is being prompted for, a template associated with the parameter type is called and used to generate the prompt. Most of these prompts are in the form of "Please say the X", where X is the parameter name.

(c) For each state in the FSM, a help prompt can be played to the user if requested or if the FSM determines it is required. For prompts where the parameter being prompted for requires a corresponding help prompt, as determined by the parameter type, a template for help prompts associated with the parameter type is used to generate the prompt. These take the form of "Please say the X", where X is the parameter name and this is immediately followed by an example, such as "For example, three hundred and twenty five dollars and fifty cents". For enumerated parameters, the first three elements in the enumeration list can be used to form the example in the prompt.

The grammar generated by the application generator 42 can be significantly enhanced to cater for a wide variety of responses. A grammatical infrared engine 46 is able to operate on response examples or observations to create additional grammars to extend the grammar 24. The examples can be provided directly to the grammatical inference engine 46 from observations recorded by the IVR. Examples can also be provided from a symbolic inference engine 48 which can generates additional examples from the predefined grammar 24. The symbolic inference engine 48 uses a linguistic and/or symbolic manipulation to extend the examples, and can rely on synonyms or antonyms extracted from a thesaurus. The symbolic inference engine 48 may also accommodate cooperative responses, which may provide additional useful information to a prompt. For instance, the prompt may be "What time would you like to travel to Melbourne?", and a cooperative response would be "I want to go to Malvern not Melbourne". Another form

of cooperative response is a preemptive response, such as to the prompt "Do you have a fax machine I could fax the information to?", the response may be "Yes my fax number is 9254 2770". Whilst a number of different grammatical inference engines could be used to extend the grammar 24, described below is a new model merging process 50 for grammatical inference that is particularly advantageous and efficient, and is executed by the engine 46.

The model merging process 50 of the grammatical inference engine 46, as shown in Figure 4, operates on the grammar 24 and additional observations. The process 50 has two distinct phases, a phrase addition phase 52 and a merging phase 54. During the phrase addition phase 52 repeated sequences of words, i.e. phrases, in the grammar that appear in more than one rule are placed by a reference to a new rule which contains the repeated phrase. For instance prior to the phrase addition phase the rules for two non-terminals may be as follows:

```
A -> b c d e
B -> x c d k
```

After the phrase addition phase 52 three rules may be defined as follows:

```
A -> b C e
B -> x C k
C -> c d
```

Above, and in the remainder of the specification, non-terminal symbols are represented by uppercase characters, whereas terminal symbols are represented by lowercase characters.

The merging phase 54 is able to merge symbols which can be considered to be equivalent. The symbols may be terminal or non-terminal. For example, if A and B are assumed to be same in the following rules,

```
X -> a A b h
Y -> q B h k
A -> y u i
B -> Z t y
```

Then after merging A and B into C the grammar would be

```
X -> a C b h
Y -> q C h k
C -> y u i
C -> Z t y
```

The symbols A and B can be merged into symbol C when the phrases are identified by merging evidence patterns, discussed below, as being interchangeable.

Merging reduces the complexity of a grammar and generalises the grammar so that it can handle additional phrases. The phrase addition process 52 does not generalise the grammar, but may create a new non-terminal that adds to the hierarchical structure of the grammar.

The model merging process 50 is able to operate on a list of rules, such as the rules of the predefined grammar 24 and rules which represent observations. A rule is assigned a probability of occurring which can be calculated from the probabilities of the observations. These probabilities can be estimated by counting the number of times the observation has occurred and dividing by the total number of observations. The merging process 50 does not create or use cost functions, which have been used by some grammatical inference engines to decide steps to be executed and when execution should cease. The model merging process 50 is based on the following principles:

(1) Whenever a sequence of two or more symbols is observed, a new non-terminal and rule are created. In representing the rule, the new non-terminal is placed on the left hand side of the rule and the observed repeated sequence of symbols on the right hand side, as shown above. All observed sequences of the symbols on the right hand side of the new rule in the grammar are replaced by the new non-terminal.

(2) The new rule needs to be applied more than once. A rule which is created during the phrase addition process is deleted, as discussed below, if it is not used more than once in parsing the observations and it is not a top-level rule. A top-level rule of a grammar has a non-terminal on the left hand side which represents the rule and resides at the highest level of the grammar.

(3) In the merging phase, merging evidence patterns, described below, are used to identify symbols used interchangeably in the observations. The process seeks two symbols used interchangeably in the same place in a sequence of words.

(4) The rules are each allocated a hyperparameter which correlates to the number of times the rule is used for the observations parsed by the generated grammar. Probabilities for the rules can then be determined from the hyperparameters. The grammatical inference engine 46 continually changes the structure of the

grammar 24, and the use of the hyperparameters significantly reduces the amount of computation required as opposed to having to calculate and store rule probabilities. A discussion on the use of hyperparameters to calculate rule probabilities for model merging is discussed in Andreas Stolcke, "Bayesian Learning of Probabilistic Language Models", 1996, Doctoral Dissertation, Department of Electrical Engineering and Computer Science, University of California, Berkeley, available at

ftp://ftp.icsi.berkeley.edu/pub/ai/Stolcke/thesis.ps.z.

The phrase addition phase 52 executes a procedure based on the Sequitur algorithm described in Neville-Manning, C.G., "Inference of Sequential Structure", 1996, Ph.D. Thesis, Department of Computer Science, University of Waikato, available at

http://sequence.rutgers.edu/~nevill/publications/Neville-Manning.pdf.

The Sequitur algorithm is used to extract a hierarchical structure from a single observation, whereas the phrase addition phase 52 is used to extract a common hierarchical structure from multiple observations.

Rules are stored using a double linked list format where each symbol has pointers to the preceding and succeeding symbol. The format enables long rules to be shortened and lengthened without computationally expensive replication of data, and a sequence of symbols can be moved from one rule to another by changing the pointers at either end, without modifying data in between.

During the phrase addition phase 52, as shown in Figure 5, observations are retrieved at step 60 and operate on one symbol at a time by retrieving the next symbol of the observation at step 62 and applying it to a newly created rule. The rule is checked for uniqueness at step 64 by executing a rule uniqueness procedure 100, as shown in Figure 7. The next symbol of the observation is then retrieved at step 66 and operation returns to step 62, unless the last symbol has been reached. If the last symbol of the observation has been reached, the next observation is retrieved at step 68 and operation returns to step 62 from step 70 until the last observation has been operated on.

Principle (1) is implemented by using a "digram" hash table. A digram is a sequence of

two consecutive symbols, where "a b" is one digram while "a c" is another. Principle (1), which relates to rule uniqueness, implies that digram within rules may be unique also. Because of this property a digram hash table is used that contains references to all instances of a particular digram. Each observation is added to the grammar in a trivial form. That is an observation O with count C(O) is added as a rule of the form S -> O where S is the top-level grammar to be operated on. The hyperparameter of the rule would be H(O).

Principle (1) is executed by the rule uniqueness procedure 100 which begins execution at step 102, as shown in Figure 7, by retrieving the first symbol in a rule. If the symbol is determined to be non-terminal at step 104, operation proceeds to step 106, otherwise another symbol is retrieved by returning to step 102 from step 108. At step 106, the entry in the digram hash table pointed to by the digram formed by the last two symbols in the rule retrieved is examined. If no such entry exists, then entry is added to the hash table pointing to the last two retrieved symbols. Operation then proceeds to step 108 to retrieve the next symbol, and if this symbol is not decided to be the last symbol at step 110, operation returns to step 104. If the last symbol has been reached after step 108, operation of the procedure 100 is completed. If an entry in the digram hash table does exist for the two symbols, then the symbols need to be replaced by a pointer to a rule containing those symbols, and a check is made at step 114 to determine if the digram table refers to an existing rule of length two, to which a reference can be made. If the rule does exist it can be reused at step 118, otherwise a new rule needs to be created at step 116 to refer to the two symbols. Once the two symbols have been replaced by a new symbol and rule, the rule is added to the observation list to check for phrase addition.

To meet principle (2) a reference counter is attached to each rule which contains the number of other rules that reference the rule, and the reference count is distinct from the hyperparameter. When a rule refers another rule, the reference count on the other rule is incremented. When a rule ceases to use another rule, the reference count on the other rule is decremented. When the reference count for a rule becomes zero, the symbols contained in that rule are placed back in the rule which previously referred to or referenced the rule that now has a reference count of zero. Accordingly, after steps 116 and 118 of the uniqueness procedure 100, the reference counts for the rules are updated at step 120, and if a reference count is determined as having reached zero at step 112, the rule is deleted at step 124. The hyperparameters and rule arrays or lists are then updated at step 126, after which operation proceeds to step 108.

Operation of the rule uniqueness procedure is illustrated with reference to the following example, where the observations are:

```
i like coffee in the morning (10)
i like tea in the morning (20)
i'd like a cup of coffee please (5)
```

The number at the end of the observations above is the number of times that phrase has been observed. In the rules below there is a two dimensional vector (i,j) at the end of the rule. The first element i of the vector is the reference count and the second element j of the vector is the hyperparameter. Processing the symbols of the observations sequentially:

Step 1.

```
S -> i (1,10)
```

Step 2.

```
S -> i like (1,10)
```

Step 3.

```
S -> i like coffee (1,10)
```

Until

Step 7.

```
S -> i like coffee in the morning (1,10)
S -> i (1,20)
```

Step 8.

```
S -> i like coffee in the morning (1,10)
S -> i like (1,20)
```

Step 9.

```
X1 -> i like (2,30)
S -> X1 coffee in the morning (1,10)
S -> X1 (1,20)
```

Step 10.

```
X1 -> i like (2,30)
```

```
S -> X1 coffee in the morning (1,10)
S -> X1 tea (1,20)
```

Step 11.

```
X1 -> i like (2,30)
S -> X1 coffee in the morning (1,10)
S -> X1 tea in (1,20)
```

Step 12.

```
X1 -> i like (2,30)
S -> X1 coffee in the morning (1,10)
S -> X1 tea in the (1,20)
```

Step 13.

```
X1 -> i like (2,30)
X2 -> in the (2,30)
S -> X1 coffee X2 morning (1,10)
S -> X1 tea X2 (1,20)
```

Step 14.

```
X1 -> i like (2,30)
X2 -> in the (2,30)
S -> X1 coffee X2 morning (1,10)
S -> X1 tea X2 morning (1,20)
```

Step 15.

```
X1 -> i like (2,30)
X2 -> in the (1,30)
X3 -> X2 morning (2,30)
S -> X1 coffee X3 (1,10)
S -> X1 tea X3 (1,20)
```

Step 16.

```
X1 -> i like (2,30)
X3 -> in the morning (2,30)
S -> X1 coffee X3 (1,10)
S -> X1 tea X3 (1,20)
```

The merging phase procedure 54, as shown in Figure 6, begins at step 130 where the

state of a variable Did_merge is checked to determine if it is "false". Did_merge is initially set to false. Next, as step 132, the first rule to be examined is retrieved and is processed at step 134 to determine when there is evidence for effecting a merger. Based on principle (3), a set of evidence patterns is established to determine when merging needs to occur. The four evidence patterns and the required merger action is recited in Table 1 below.

Evidence	Action
X -> A B X -> A C	Merge B and C X -> A Y Y -> B Y -> C
X -> A B X -> C B	Merge A and C X -> Y B Y -> A Y -> C
X -> A B C X -> A D C	Merge B and D X -> A Y C Y -> B Y -> D
X -> A D X -> F E D	Merge A and F E X -> Y D Y -> A Y -> F E
X -> A B X -> A C O	Merge B and C O X -> A Y Y -> B Y -> C O
X -> Y	Merge X and Y, where both are non-terminals Delete evidence rule

Table 1

For all of the actions described in the above table, with the exception of the last action involving deleting the evidence rule, the symbols which are merged may be non-terminals or terminals. If a symbol to be merged is a non-terminal, then it is not necessary to create a rule of the form Y -> A where only one symbol is on the right hand side. A rule Y -> a only needs to be created for terminals involved in the merger. If, for example, all of the symbols to be merged are non-terminals, then the symbols can be replaced on the right hand side and left hand side of

the rules of the grammar in which they appear with a reference to a new non-terminal, and no additional rules need to be created.

If there is evidence for the merge, the merger is executed, as explained further below, at step 136 and Did_merge is set to "true" at step 138. Any redundant rules are deleted at step 140, and the rule created as a result of the merger is checked for uniqueness by procedure 64. Operation then proceeds to step 144 where a determination is made as to whether the last rule has been processed. If not, an iterator that points to the rules is incremented at step 146 and operation returns to step 134. If the last rule has been reached, a check is made to determine whether any merger has occurred by examining whether Did_merge is false. If it is true, operation returns to step 130, otherwise if it is false, the procedure ends.

Continuing on from the previous example after the end of the phase addition phase 52 the grammar will be as follows:

```
X1 -> i like (2,30)
X3 -> in the morning (2,30)
S -> X1 coffee X3 (1,10)
S -> X1 tea X3 (1,20)
S -> i'd like a cup of coffee please (1,5)
```

Using the four different merging patterns listed in Table 1, the grammar is altered as follows:

```
Step  18 (merge coffee and tea)
X1 -> i like (2,30)
X3 -> in the morning (2,30)
X4 -> coffee ( 2,15)
X4 -> tea ( 1,20)
S -> X1 X4 X3 (1,10)
S -> X1 X4 X3 (1,20)
S -> i'd like a cup of X4 please (1,5)

Step  19
X1 -> i like (2,30)
X3 -> in the morning (2,30)
X4 -> coffee ( 2,15)
X4 -> tea ( 1,20)
S -> X1 X4 X3 (1,30)
S -> i'd like a cup of X4 please (1,5)
```

Using the doubly linked list structure, only the symbol on the left hand side needs to be changed for all of the rules that use it. In the example given "coffee" and "tea" are terminals. If one of the symbols to be merged is a terminal a rule is created with that symbol on the right hand side.

Principle (4) is satisfied by the phase addition and merge procedures 52 and 54 adjusting the hyperparameters. When a rule for a phrase is added, as in Step 9 of the example the hyperparameter of a rule is equal to the sum of the hyperparameters of the two rules that use it. When a phrase rule is added that uses an existing rule, the hyperparameter of the existing rule is increased by the hyperparameter of the new rule. When two rules are merged, the hyperparameter of the newly formed rule is the sum of the two previous hyperparameters as in Step 19 of the example.

After the model merging procedure is completed the probabilities of the rules can be calculated

```
X1 -> i like (30/30)
X3 -> in the morning (30/30)
X4 -> coffee (15/35)
X4 -> tea (20/35)
S -> X1 X4 X3 (30/35)
S -> i'd like a cup of X4 please (5/35)
```

It can be seen that the new grammar is more general than the observations. For instance it can generate the phrase

```
i'd like a cup of tea please
```

According to the inferred grammar the probability of this phrase is calculated to be $5/35 * 20/35 \sim 0.08$.

Principles (1) and (2) continue to be applied during the merging phase 54. Merging nearly always reduces the number of rules as whenever a merge is identified either two rules are merged or one is deleted. When two rules are merged, one rule is deleted and the hyperparameter of one is increased by the hyperparameter of the deleted rule.

In order to operate well, the model merging procedure 50 uses as a data structure to

ensure the following operations are executed with efficiency:

(a) Appending a symbol to a rule.

(b) Using an existing rule.

(c) Creating a new rule.

(d) Deleting a rule.

(e) Merging rules.

(f) Iterating through the rules.

In the data structure:

(i) Each word is replaced by a single integer representing the word.

(ii) Each symbol is represented by a structure that contains a symbol identification (id) number, plus a pointer to the previous and subsequent symbol in a rule. Each rule has guard symbol Ω which points to both the first and the last symbols in the rule. This implements the doubly linked list.

(iii) Each rule is attached to a structure representing a non-terminal.

The data structure and its use is illustrated in Figures 8 to 12 which respectively show how the data structure changes for appending a symbol, reusing an existing rule, creating a rule, deleting a rule and merging two symbols B and D into symbol Z.

All non-terminals are referenced in a global associative array, which associates the non-terminal id with the non-terminal. If states A and B are to be merged, then all non-terminal structures referenced in the array are accessed and the non-terminal id on each of these is changed. Because the symbol id of a non-terminal is determined by the non-terminal it addresses, all occurrences on the right hand side of all rules are automatically modified. All relevant associative arrays and hash tables are then updated, as in step 126 of Figure 7.

The list of rules are iterated through, by the use of a pointing iterator, as shown in Figure 13. The rules are iterated through in a re-entrant fashion. In this context re-entrant means that a rule could be deleted, and the remaining rules could be iterated through without affect. This is achieved by storing the rules in the global doubly linked list format. As shown in Figure 13, initially the iterator points to rule 1, and then subsequently is incremented to point to rule 2. Then if rule 2 is to be deleted, prior to deletion of the rule, the rule is removed from the doubly

linked list, and then the iterator is incremented to point to rule 3.

Importantly the model merging process 50 is able to include predefined grammars. Examples of this are the grammar generated by application generator 42 and a grammar that defines numbers or dates. This can be achieved simply adding the rules to the grammar, and applying principles (1), (2) and (4) prior to the start of the phrase addition phase 52. If the predefined grammar is in a suitable format minimal processing is required to do this. In the general case rules with a single symbol on the right hand side may exist, and thus an additional associative array is required, so that during the phrase addition phase 52 all symbols of type x can be replaced by the non-terminal X where a rule of the form X -> x exists.

Another advantageous method of integrating the output of the application generator 42 and the model merging process 50 is to define a list of non-terminals created in the grammar that represent the predefined parameter types and non-terminals created for each enumerated item from the application file 44. Observations are then partially parsed to replace those portions of the observation that can be parsed by the grammar with the appropriate non-terminals. For instance, using the application file 44 of Appendix 1 and the generated grammar 24 of Appendix 4, the non-terminals would be INTEGER, money and Stockname. The phrase "I would like to buy three hundred shares of york securities" would be partially parsed to become "I would like to buy INTEGER shares of Stockname". This partial parsing could be achieved using a known chart parser. A grammar can then be created by the model merging process 50 using the partially parsed observations as the input observations to the model merging process 50, but without using the generated grammar as the starting grammar. The grammar that is generated using the model merging process 50 is then added to the grammar 24 generated using the application generator 42. The advantage of using this technique instead of using the generated grammar as the starting grammar to the model merging process is that the structure that exists in the grammar relates to the meaning of the rules, rather than just the syntax.

Another important feature of the model merging procedure 50 is where the generated grammar cannot be recursive. To overcome this the rules are added to the grammar in the form

```
S -> X
X -> rhs
```

Rather than in the form

```
S -> rhs
```

During the merging phase 54 the merge is tested to see if it generates a recursive grammar. If it does the merge is reversed. Grammar can be tested for recursion using a number of techniques. A preferred method involves execution of a recursive procedure 200, as shown in Figure 14 which sets a variable current non-terminal to be the top-level grammar at step 202 and then calls a traverse node procedure 250, as shown in Figure 15, which initially operates on an empty list of non-terminals. The procedure 250 is used recursively to test all of the rules in the grammar that have a particular non-terminal on the left hand side of a rule. A list of non-terminals that have been previously checked is passed to the procedure 250 when subsequently called. The current non-terminal is added to the list of previously tested rules at step 252 and at step 254 the next rule with the current non-terminal on the left hand side is accessed. A test is then executed at step 256 to determine if the last rule has been reached. If yes then a variable recursive is set to false at step 258 and the procedure completes. Otherwise if the last rule has not been reached then the next symbol in the rule is retrieved at step 260, and a determination made at step 60 to see whether the symbol exists in the symbol list, i.e. if a non-terminal appearing on the right hand side already exists in the list of previously tested non-terminals. If so, the variable recursive is set to true at step 264 and the procedure completes. If not, another instance of the procedure 250 is executed on the non-terminal. When an instance of the procedure 250 generated in this manner completes a test is made at step 266 to determine if the variable recursive is true. If so, the procedure completes, otherwise a test is then made to determine whether the last symbol in the non-terminal list has been reached at step 268. If not, operation returns to step 260, otherwise operation will return to step 254 to retrieve the next rule.

To illustrate how the procedures 200 and 250 operate, Table 2 below sets out the sequence of the rules examined and the compilation of the non-terminal list for the following grammar:

```
S -> a X1 b
S -> d e
X1 -> a X2 b
X1 -> a S b
X2 -> i
```

	Rule Being Examined	Non-Terminal List
(i)	S -> a X1 B	S
(ii)	X1 -> a X2 b	S,X1
(iii)	X2 -> i	S,X1,X2
(iv)	X1 -> a S b	S,X1 Grammar is recursive

Table 2

The table shows that the top-level grammar S is checked first. When checking the first rule, the non-terminal list contains only the symbol S, and when the symbol X1 is checked in the first rule, the procedure 250 is called recursively to check if any of the rules attached to X1 are recursive, and the symbol X1 is added to the list. The symbol X2 is encountered and also added to the list. When the symbol S is encountered in the right hand side of the second rule for X1, the procedure 250 identifies at step 262 that this symbol already exists in the list, and the grammar is then identified as recursive and the procedure completes. When a test is executed to determined whether a grammar is recursive, and the grammar is known not to be recursive prior to executing a merge, the test for recursion can start with the newly merged non-terminal rather than the top-level grammar. This will reduce execution time when the newly merged non-terminal is not a top-level non-terminal.

All of the processes and components of the development system 40 are preferably executed by a computer program or programs. Alternatively the processes and components may be wholly or partly implemented in hardware, such as by ASICs. The components may also be distributed or located in one location.

Many modifications will be apparent to those skilled in the art without departing from the scope of the present invention as hereinbefore described with reference to the accompanying drawings.

DATED this 15th day of December, 1999
TELSTRA R&D MANAGEMENT PTY. LTD.
By its Patent Attorneys
DAVIES COLLISON CAVE

APPENDIX 1

```
<?xml version='1.0' encoding='utf-8' ?>
<APPLICATION NAME="Stock">
    <ENUMERATED NAME="stockname">
        <ITEM NAME="philp burns"/>
        <ITEM NAME="macquarie qanmacs"/>
        <ITEM NAME="a i engineering"/>
        <ITEM NAME="a p eagers limited"/>
        <ITEM NAME="a a p c limited "/>
        <ITEM NAME="a a p t limited"/>
        <ITEM NAME="abador gold "/>
        <ITEM NAME="abednego nickel"/>
        <ITEM NAME="aberfoyle limited"/>
        <ITEM NAME="abigroup limited"/>
        <ITEM NAME="arthur yates "/>
        <ITEM NAME="york securities"/>
        <ITEM NAME="zeolite australia"/>
        <ITEM NAME="zephyr minerals "/>
        <ITEM NAME="zicom australia"/>
        <ITEM NAME="zimbabwe platinum"/>
        <ITEM NAME="zylotech limited"/>
    </ENUMERATED>

    <OPERATION NAME="buy" >
        <PARAMETER NAME="stockname" TYPE="stockname"/>
        <PARAMETER NAME="number" TYPE="INTEGER"/>
        <PARAMETER NAME="price" TYPE="money"/>
    </OPERATION>
    <OPERATION NAME="sell" CONFIRM="yes">
        <PARAMETER NAME="stockname" TYPE="stockname"/>
        <PARAMETER NAME="number" TYPE="INTEGER"/>
        <PARAMETER NAME="price" TYPE="money"/>
    </OPERATION>
    <OPERATION NAME="quote" CONFIRM= "yes">
        <PARAMETER NAME="stockname" TYPE="stockname"/>
      <RETURN NAME="stockname" TYPE="stockname"/>
      <RETURN NAME="date" TYPE="date"/>
      <RETURN NAME="price" TYPE="money"/>
    </OPERATION>
</APPLICATION>
```

APPENDIX 2

```
PROCESS Stock;
  DCL buystockname STOCKNAME, buynumber INTEGER, buyprice MONEY,
    sellstockname STOCKNAME, sellnumber INTEGER, sellprice MONEY,
    quotestockname STOCKNAME, quotedate DATE, quoteprice MONEY;
  START;
    OUTPUT TopLevelStock;

    NEXTSTATE TopLevelStock;

  STATE TopLevelStock;
    INPUT buy(stockname, number, price);
      TASK buystockname = stockname;
      TASK buynumber = number;
      TASK buyprice = price;
      CALL Askbuystockname;
      CALL Askbuynumber;
      CALL Askbuyprice;
      CALL Completebuy;
      NEXTSTATE TopLevelStock;

    INPUT sell(stockname, number, price);
      TASK sellstockname = stockname;
      TASK sellnumber = number;
      TASK sellprice = price;
      CALL Asksellstockname;
      CALL Asksellnumber;
      CALL Asksellprice;
      CALL ConfirmCompletesell;
      NEXTSTATE TopLevelStock;

    INPUT quote(stockname);
      TASK quotestockname = stockname;
      CALL Askquotestockname;
      CALL Completequote;
      NEXTSTATE TopLevelStock;
  STATE *;
  /* This applies to all states */
    INPUT cancel;
      OUTPUT operationcancelled;
      NEXTSTATE TopLevel;

    INPUT help;
      OUTPUT helpstate*;
      /* The name of the state is appended to
      the prompt name */
      NEXTSTATE -;
      /* return to the same state */

    INPUT quit;
      OUTPUT quit;
      STOP;

ENDPROCESS Stock;

PROCEDURE Askbuystockname;
  START;
    DECISION buystockname;
      (= ""):
        OUTPUT Askbuystockname;
```

```
        NEXTSTATE WaitReplyAskbuystockname;

      (/= ""):
        RETURN;

    ENDDECISION;

  STATE WaitReplyAskbuystockname;
    INPUT buystockname(stockname, number, price);
      CALL setbuystocknameIfNotNil(stockname);
      CALL setbuynumberIfNotNil(number);
      CALL setbuypriceIfNotNil(price);
      RETURN;

ENDPROCEDURE Askbuystockname;

PROCEDURE setbuystocknameIfNotNil;
  FPAR IN/OUT stockname STOCKNAME;
  START;
    DECISION stockname;
      (= ""):
        RETURN;

      (/= ""):
        TASK buystockname = stockname;

    ENDDECISION;

ENDPROCEDURE setbuystocknameIfNotNil;

PROCEDURE Askbuynumber;
  START;
    DECISION buynumber;
      (= ""):
        OUTPUT Askbuynumber;
        NEXTSTATE WaitReplyAskbuynumber;

      (/= ""):
        RETURN;

    ENDDECISION;

  STATE WaitReplyAskbuynumber;
    INPUT buynumber(number, price);
      CALL setbuynumberIfNotNil(number);
      CALL setbuypriceIfNotNil(price);
      RETURN;

ENDPROCEDURE Askbuynumber;

PROCEDURE setbuynumberIfNotNil;
  FPAR IN/OUT number INTEGER;
  START;
    DECISION number;
      (= ""):
        RETURN;

      (/= ""):
        TASK buynumber = number;

    ENDDECISION;
```

```
ENDPROCEDURE setbuynumberIfNotNil;

PROCEDURE Askbuyprice;
  START;
    DECISION buyprice;
      (= ""):
        OUTPUT Askbuyprice;
        NEXTSTATE WaitReplyAskbuyprice;

      (/= ""):
        RETURN;

    ENDDECISION;

  STATE WaitReplyAskbuyprice;
    INPUT buyprice(price);
      CALL setbuypriceIfNotNil(price);
      RETURN;

ENDPROCEDURE Askbuyprice;

PROCEDURE setbuypriceIfNotNil;
  FPAR IN/OUT price MONEY;
  START;
    DECISION price;
      (= ""):
        RETURN;

      (/= ""):
        TASK buyprice = price;

    ENDDECISION;

ENDPROCEDURE setbuypriceIfNotNil;

PROCEDURE Completebuy;
  START;
    CALL buy(buystockname, buynumber, buyprice);

    OUTPUT buyConfirmed;

    NEXTSTATE WaitConfirmbuyEnd;

  STATE WaitConfirmbuyEnd;
    INPUT continue;
      RETURN;

ENDPROCEDURE Completebuy;

PROCEDURE Asksellstockname;
  START;
    DECISION sellstockname;
      (= ""):
        OUTPUT Asksellstockname;
        NEXTSTATE WaitReplyAsksellstockname;

      (/= ""):
        RETURN;

    ENDDECISION;

  STATE WaitReplyAsksellstockname;
```

```
    INPUT sellstockname(stockname, number, price);
      CALL setsellstocknameIfNotNil(stockname);
      CALL setsellnumberIfNotNil(number);
      CALL setsellpriceIfNotNil(price);
      RETURN;

ENDPROCEDURE Asksellstockname;

PROCEDURE setsellstocknameIfNotNil;
  FPAR IN/OUT stockname STOCKNAME;
  START;
    DECISION stockname;
      (= ""):
        RETURN;

      (/= ""):
        TASK sellstockname = stockname;

    ENDDECISION;

ENDPROCEDURE setsellstocknameIfNotNil;

PROCEDURE Asksellnumber;
  START;
    DECISION sellnumber;
      (= ""):
        OUTPUT Asksellnumber;
        NEXTSTATE WaitReplyAsksellnumber;

      (/= ""):
        RETURN;

    ENDDECISION;

  STATE WaitReplyAsksellnumber;
    INPUT sellnumber(number, price);
      CALL setsellnumberIfNotNil(number);
      CALL setsellpriceIfNotNil(price);
      RETURN;

ENDPROCEDURE Asksellnumber;

PROCEDURE setsellnumberIfNotNil;
  FPAR IN/OUT number INTEGER;
  START;
    DECISION number;
      (= ""):
        RETURN;

      (/= ""):
        TASK sellnumber = number;

    ENDDECISION;

ENDPROCEDURE setsellnumberIfNotNil;

PROCEDURE Asksellprice;
  START;
    DECISION sellprice;
      (= ""):
        OUTPUT Asksellprice;
        NEXTSTATE WaitReplyAsksellprice;
```

```
      (/= ""):
        RETURN;

    ENDDECISION;

  STATE WaitReplyAsksellprice;
    INPUT sellprice(price);
      CALL setsellpriceIfNotNil(price);
      RETURN;

ENDPROCEDURE Asksellprice;

PROCEDURE setsellpriceIfNotNil;
  FPAR IN/OUT price MONEY;
  START;
    DECISION price;
      (= ""):
        RETURN;

      (/= ""):
        TASK sellprice = price;

    ENDDECISION;

ENDPROCEDURE setsellpriceIfNotNil;

PROCEDURE ConfirmCompletesell;
  START;
    OUTPUT AskConfirmsell(sellstockname, sellnumber, sellprice);

    NEXTSTATE WaitAskConfirmsell;

  STATE WaitAskConfirmsell;
    INPUT yes;
      CALL sell(sellstockname, sellnumber, sellprice);
      OUTPUT sellConfirmed;
      NEXTSTATE WaitAskConfirmsellEnd;

    INPUT no;
      OUTPUT sellCancelled;
      NEXTSTATE WaitAskConfirmsellEnd;

  STATE WaitAskConfirmsellEnd;
    INPUT continue;
      RETURN;

ENDPROCEDURE ConfirmCompletesell;

PROCEDURE Askquotestockname;
  START;
    DECISION quotestockname;
      (= ""):
        OUTPUT Askquotestockname;
        NEXTSTATE WaitReplyAskquotestockname;

      (/= ""):
        RETURN;

    ENDDECISION;

  STATE WaitReplyAskquotestockname;
    INPUT quotestockname(stockname);
```

```
      CALL setquotestocknameIfNotNil(stockname);
      RETURN;

ENDPROCEDURE Askquotestockname;

PROCEDURE setquotestocknameIfNotNil;
  FPAR IN/OUT stockname STOCKNAME;
  START;
    DECISION stockname;
      (= ""):
        RETURN;

      (/= ""):
        TASK quotestockname = stockname;

    ENDDECISION;

ENDPROCEDURE setquotestocknameIfNotNil;

PROCEDURE Completequote;
  START;
    CALL quote(quotestockname, quotedate, quoteprice );

    OUTPUT quoteConfirmed(quotestockname, quotedate, quoteprice );

    NEXTSTATE WaitConfirmquoteEnd;

  STATE WaitConfirmquoteEnd;
    INPUT continue;
      RETURN;

ENDPROCEDURE Completequote;
```

APPENDIX 3

```
/* Clips output */

  (deffacts Stock-questions
    (question
      (name "Askbuystockname")
      (interactiontype "ask")
      (prompt "Please say the stockname")
      (grammar ".WaitAskbuystockname")
    )
    (question
      (name "helpstateAskbuystockname")
      (interactiontype "ask")
      (prompt "Please say the stockname.  For example, b h p, commonwealth
bank  You can say cancel to go to the top level.")
      (grammar ".WaitAskbuystockname")
    )
    (question
      (name "Askbuynumber")
      (interactiontype "ask")
      (prompt "Please say the number")
      (grammar ".WaitAskbuynumber")
    )
    (question
      (name "helpstateAskbuynumber")
      (interactiontype "ask")
      (prompt "Please say the number.  For example, three hundred and twenty
five  You can say cancel to go to the top level.")
      (grammar ".WaitAskbuynumber")
    )
    (question
      (name "Askbuyprice")
      (interactiontype "ask")
      (prompt "Please say the price")
      (grammar ".WaitAskbuyprice")
    )
    (question
      (name "helpstateAskbuyprice")
      (interactiontype "ask")
      (prompt "Please say the price.  For example, three dollars and fifty
cents.  You can say cancel to go to the top level.")
      (grammar ".WaitAskbuyprice")
    )
    (question
      (name "buyConfirmed")
      (interactiontype "tell")
      (prompt "buy operation confirmed")
      (grammar ".none")
    )
    (question
      (name "Asksellstockname")
      (interactiontype "ask")
      (prompt "Please say the stockname")
      (grammar ".WaitAsksellstockname")
    )
    (question
      (name "helpstateAsksellstockname")
      (interactiontype "ask")
      (prompt "Please say the stockname.  For example, b h p, commonwealth
bank  You can say cancel to go to the top level.")
```

```
      (grammar ".WaitAsksellstockname")
    )
    (question
      (name "Asksellnumber")
      (interactiontype "ask")
      (prompt "Please say the number")
      (grammar ".WaitAsksellnumber")
    )
    (question
      (name "helpstateAsksellnumber")
      (interactiontype "ask")
      (prompt "Please say the number.  For example, three hundred and twenty
five  You can say cancel to go to the top level.")
      (grammar ".WaitAsksellnumber")
    )
    (question
      (name "Asksellprice")
      (interactiontype "ask")
      (prompt "Please say the price")
      (grammar ".WaitAsksellprice")
    )
    (question
      (name "helpstateAsksellprice")
      (interactiontype "ask")
      (prompt "Please say the price.  For example, three dollars and fifty
cents.  You can say cancel to go to the top level.")
      (grammar ".WaitAsksellprice")
    )
    (question
      (name "AskConfirmsell")
      (interactiontype "ask")
      (prompt "According to the system you want to sell.  The stockname is
*stockname* . The number is *number* . The price is *price* . Is this
correct?")
      (grammar ".WaitAskConfirmsell")
    )
    (question
      (name "helpstateAskConfirmsell")
      (interactiontype "ask")
      (prompt "Please say yes or no.  You can say cancel to go to the top
level.")
      (grammar ".WaitAskConfirmsell")
    )
    (question
      (name "sellCancelled")
      (interactiontype "tell")
      (prompt "sell operation cancelled")
      (grammar ".none")
    )
    (question
      (name "sellConfirmed")
      (interactiontype "tell")
      (prompt "sell operation confirmed")
      (grammar ".none")
    )
    (question
      (name "TopLevelStock")
      (interactiontype "ask")
      (prompt "Welcome to the Stock application.  Please say one of the
following buy, sell, quote")
      (grammar ".TopLevelStock")
    )
```

```
    (question
      (name "helpstateTopLevelStock")
      (interactiontype "ask")
      (prompt "Please say one of the following options buy, sell, quote")
      (grammar ".TopLevelStock")
    )
(question
        (name "operationcancelled")
        (interactiontype "ask")
       (prompt "operation cancelled. Please say one of the following options
buy, sell, quote")
        (grammar ".TopLevelStock")
    )
    (question  ( name quit)
        ( interactiontype "end")
        ( prompt "goodbye")
    )

  )
```

APPENDIX 4

```
CommonStockCmds [
  CancelActions {<operation cancel>}
  RepeatAction {<operation repeat>}
  ShortHelpActions {<operation help>}
  QuitActions {<operation quit>}
]

Stockname [
  (philp burns) {return("philp burns")}
  (macquarie qanmacs) {return("macquarie qanmacs")}
  (a i engineering) {return("a i engineering")}
  (a p eagers limited) {return("a p eagers limited")}
  (a a p c limited ) {return("a a p c limited ")}
  (a a p t limited) {return("a a p t limited")}
  (abador gold ) {return("abador gold ")}
  (abednego nickel) {return("abednego nickel")}
  (aberfoyle limited) {return("aberfoyle limited")}
  (abigroup limited) {return("abigroup limited")}
  (arthur yates ) {return("arthur yates ")}
  (york securities) {return("york securities")}
  (zeolite australia) {return("zeolite australia")}
  (zephyr minerals ) {return("zephyr minerals ")}
  (zicom australia) {return("zicom australia")}
  (zimbabwe platinum) {return("zimbabwe platinum")}
  (zylotech limited) {return("zylotech limited")}
]

.WaitAskbuystockname [
  CommonStockCmds
  (stockname:x) { <operation buystockname> <stockname $x>}
]

.WaitAskbuynumber [
  CommonStockCmds
  (INTEGER:x) { <operation buynumber> <number $x>}
]

.WaitAskbuyprice [
  CommonStockCmds
  (money:x) { <operation buyprice> <price $x>}
]

GR_buy [
  buyNoun
]

BuyNoun [
  buy
]

.WaitAsksellstockname [
  CommonStockCmds
  (stockname:x) { <operation sellstockname> <stockname $x>}
]

.WaitAsksellnumber [
  CommonStockCmds
  (INTEGER:x) { <operation sellnumber> <number $x>}
]
```

```
.WaitAsksellprice [
  CommonStockCmds
  (money:x) { <operation sellprice> <price $x>}
]

GR_sell [
  sellNoun
]

sellNoun [
  sell
]

.WaitAskConfirmsell [
  CommonStockCmds
  LOOSE_CONFIRMATION:c {<operation $c>}
]

.TopLevelStock [
  CommonStockCmds
  GR_buy {<operation buy>}
  GR_sell {<operation sell>}
]
```

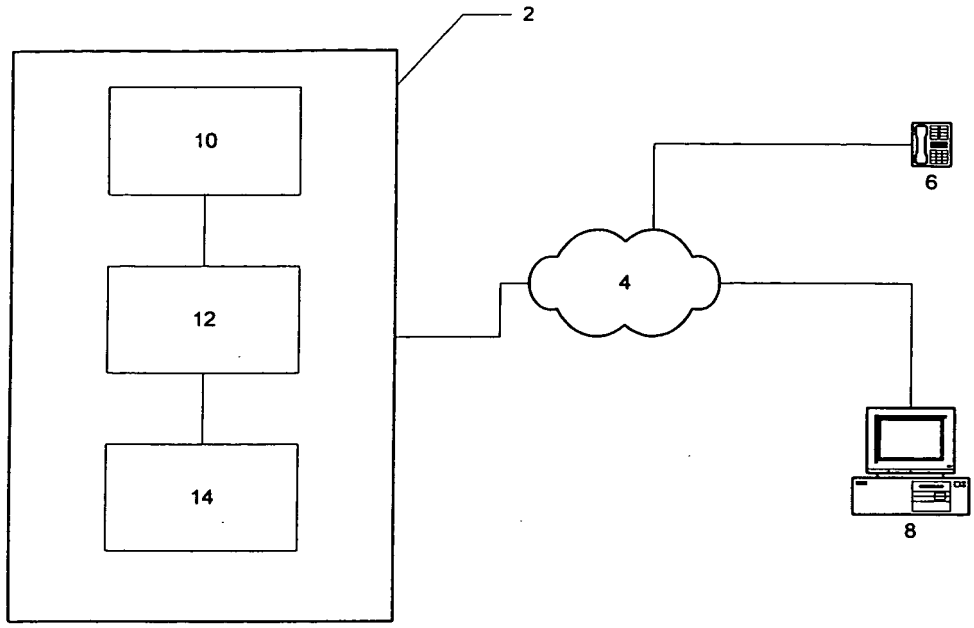


Figure 1

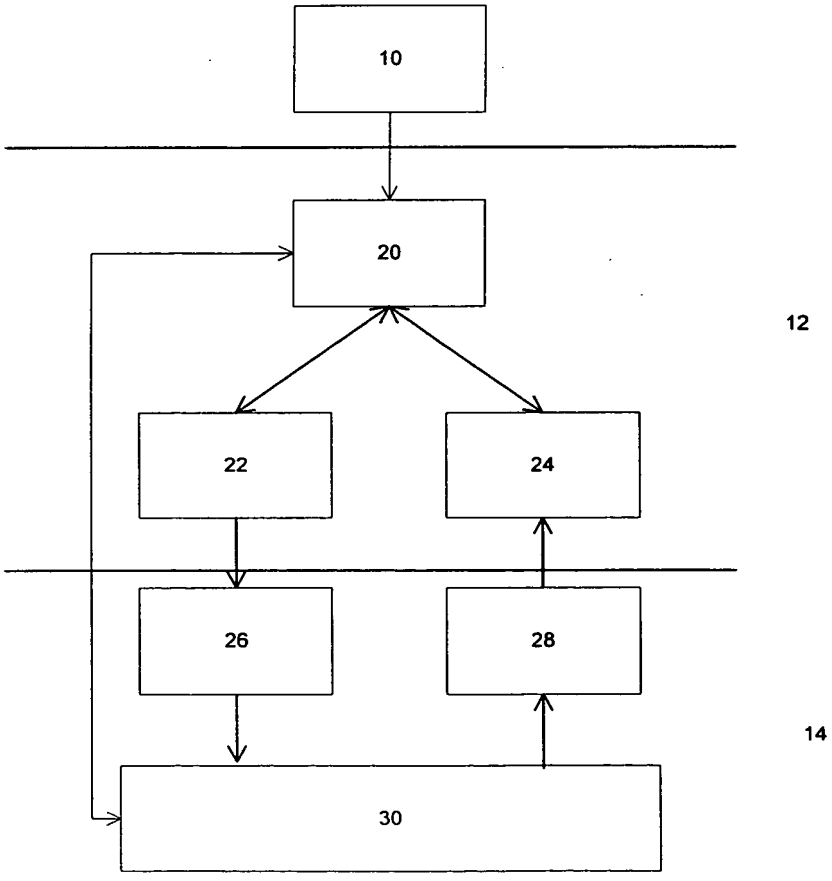


Figure 2

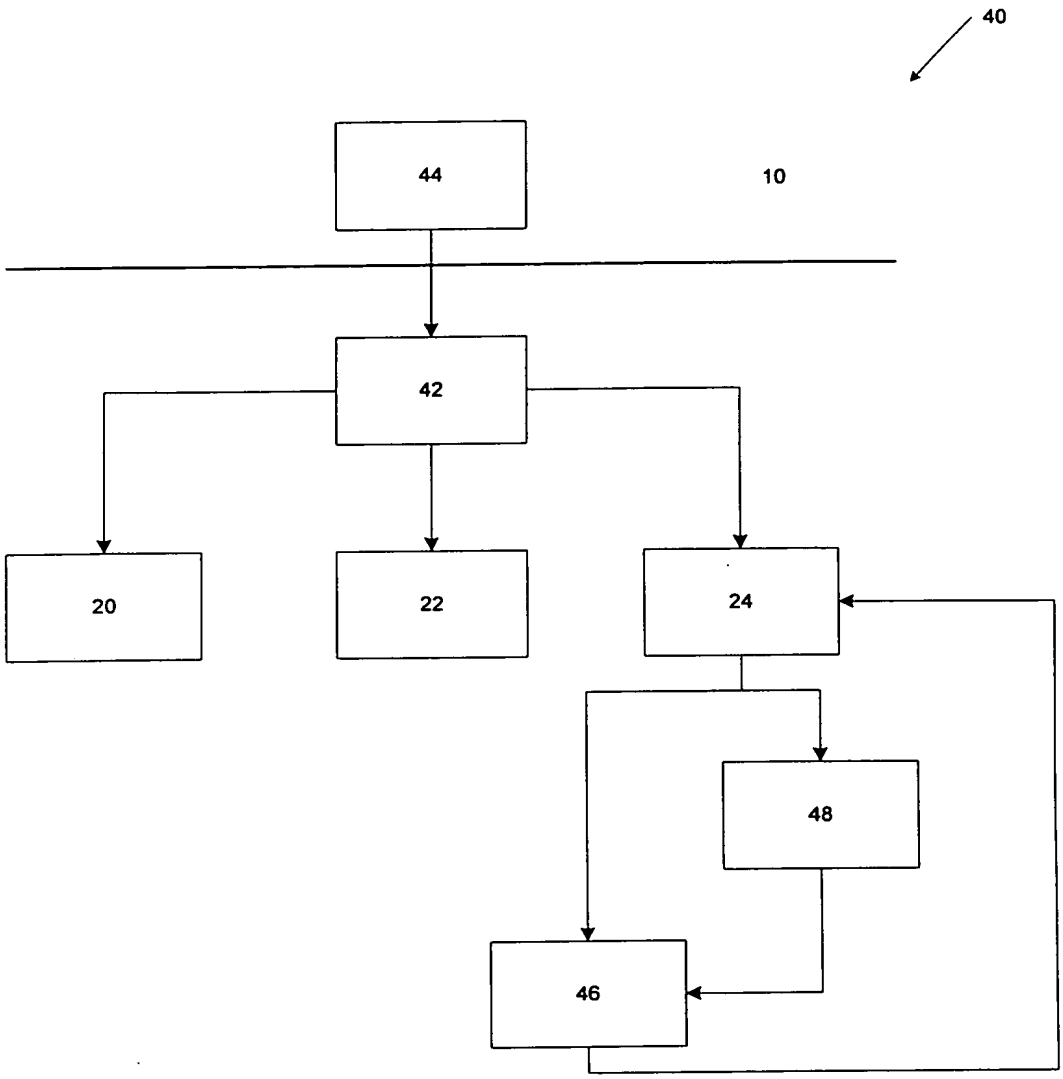


Figure 3

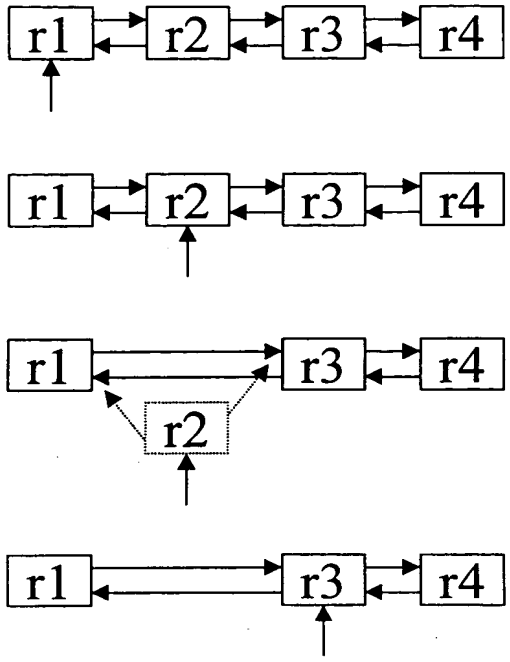


FIGURE 13

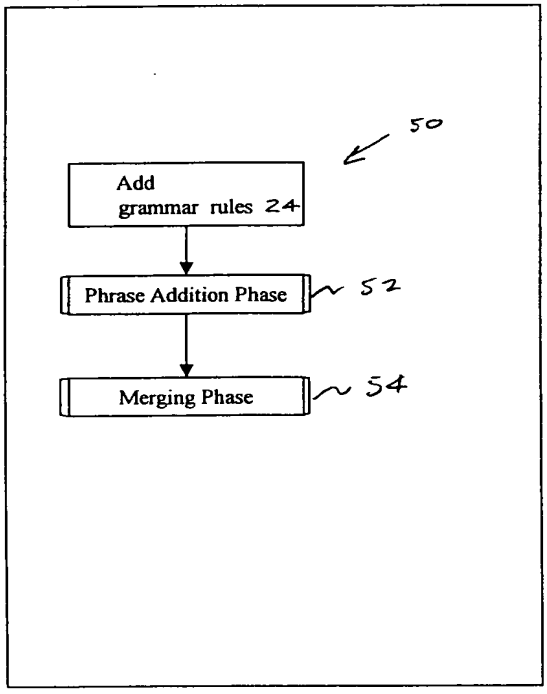


FIGURE 4

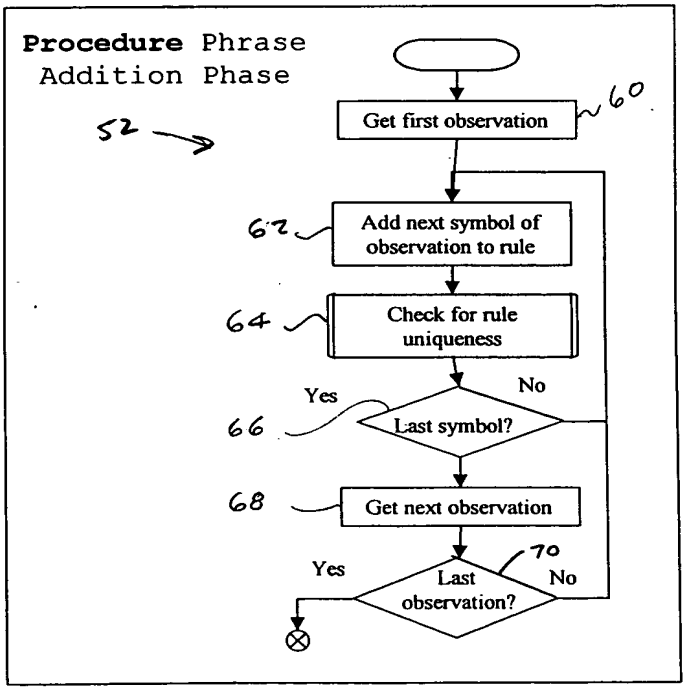


FIGURE 5

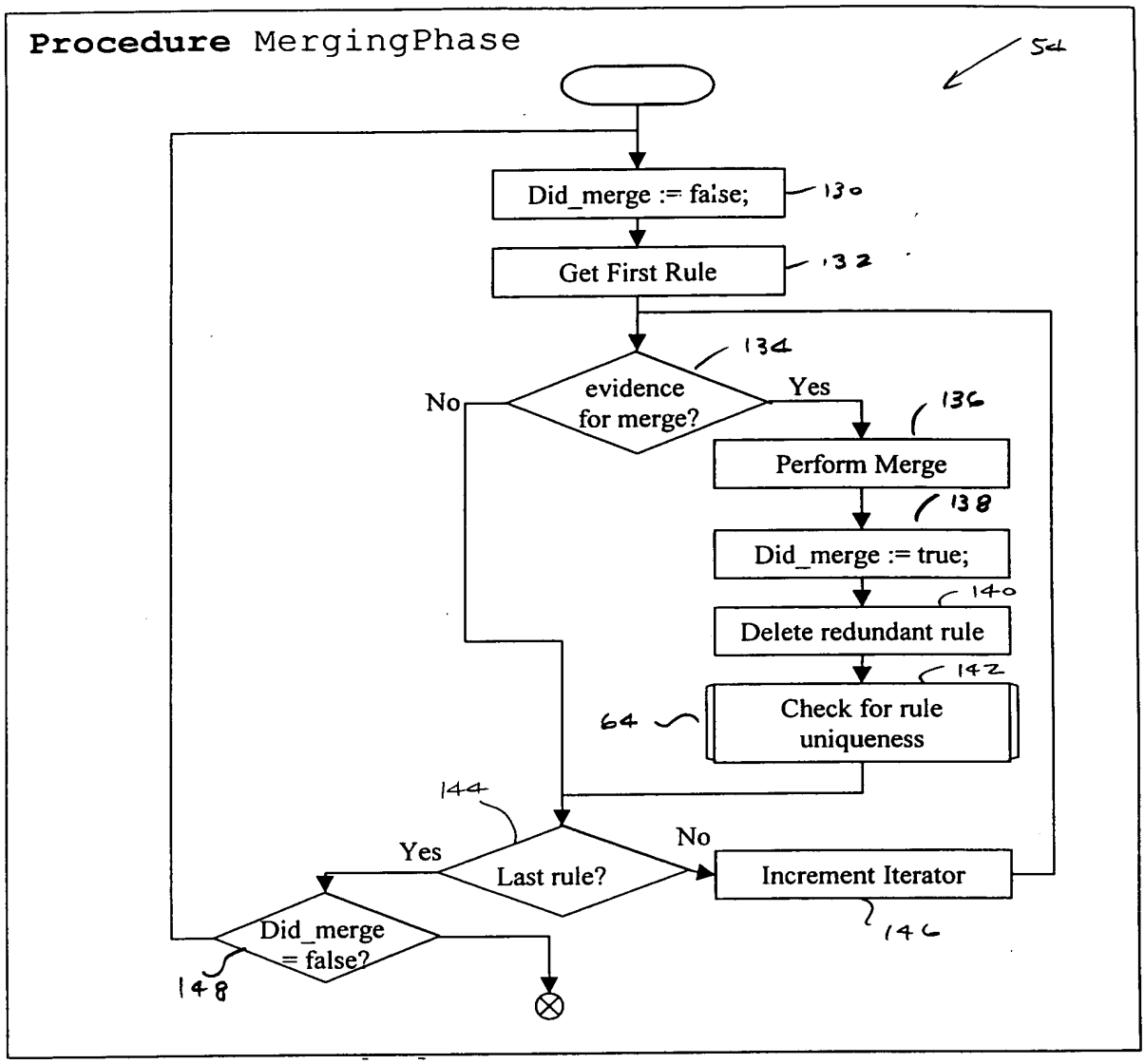

Procedure MergingPhase
54
Did_merge := false;
130
Get First Rule
132
134
evidence for merge?
No
Yes
136
Perform Merge
138
Did_merge := true;
140
Delete redundant rule
142
64
Check for rule uniqueness
144
Yes
Last rule?
No
Increment Iterator
146
Did_merge = false?
148

FIGURE 6

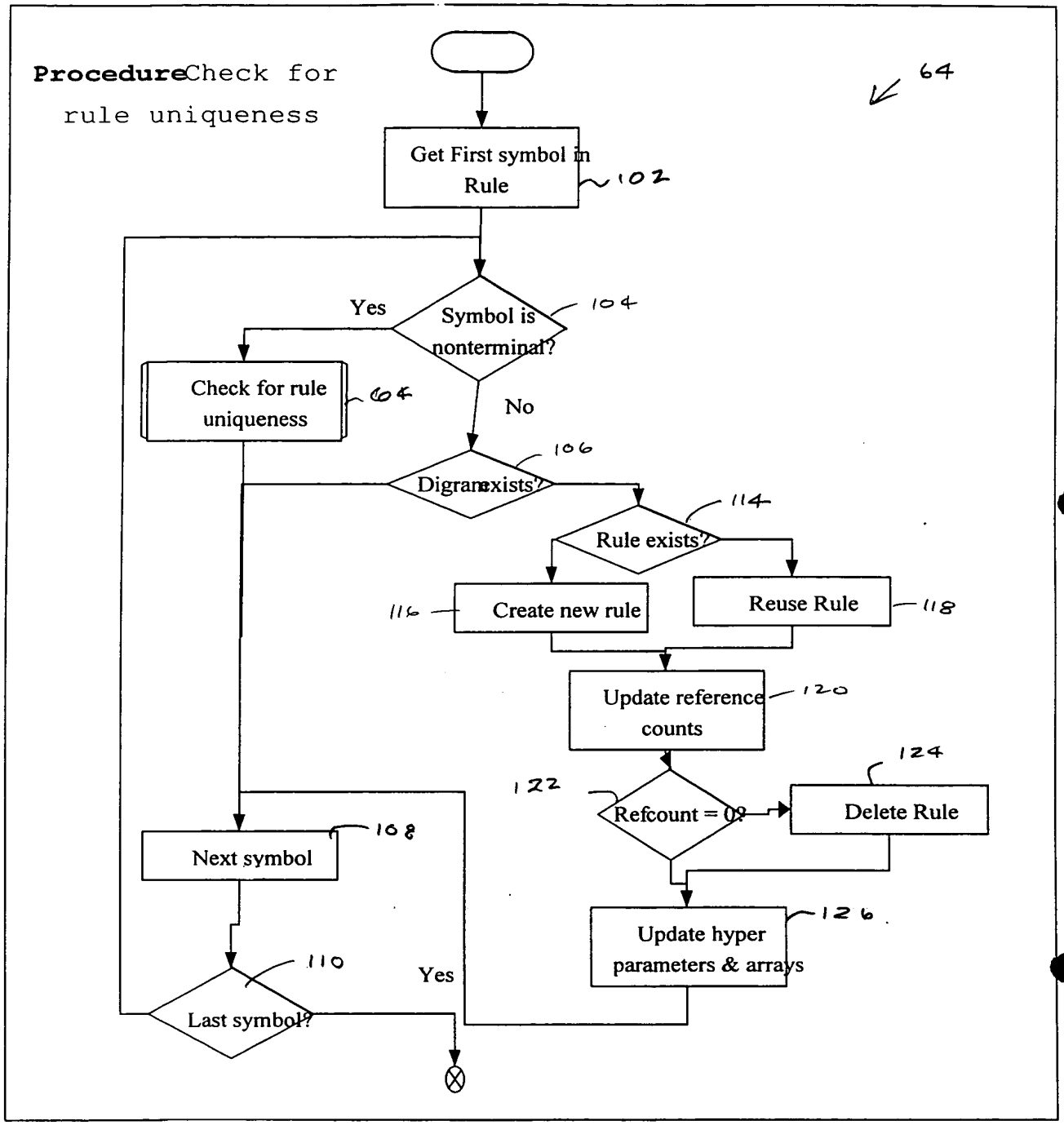

Procedure Check for rule uniqueness
64
Get First symbol in Rule
102
Yes
Symbol is nonterminal?
104
Check for rule uniqueness
64
No
106
Digram exists?
114
Rule exists?
116
Create new rule
Reuse Rule
118
Update reference counts
120
122
Refcount = 0?
124
Delete Rule
108
Next symbol
126
Update hyper parameters & arrays
110
Yes
Last symbol?

FIGURE 7

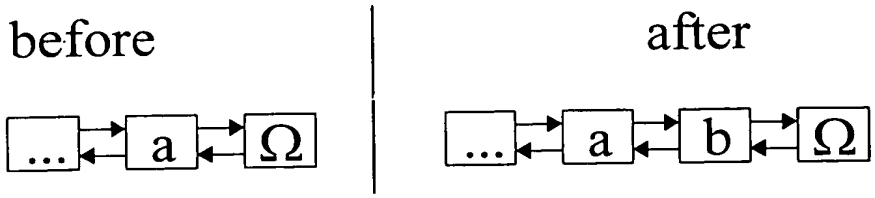


Figure 8

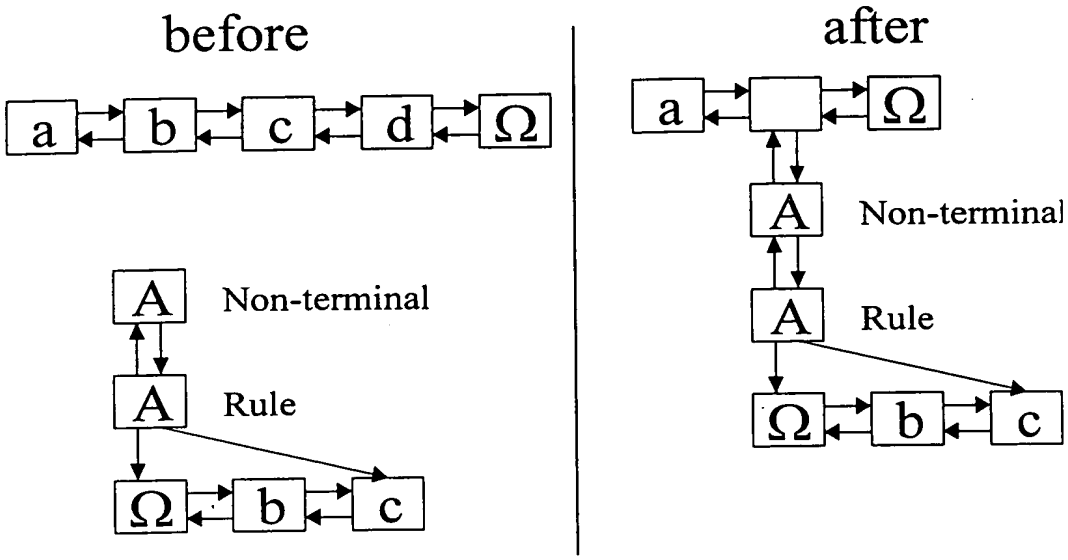


Figure 9

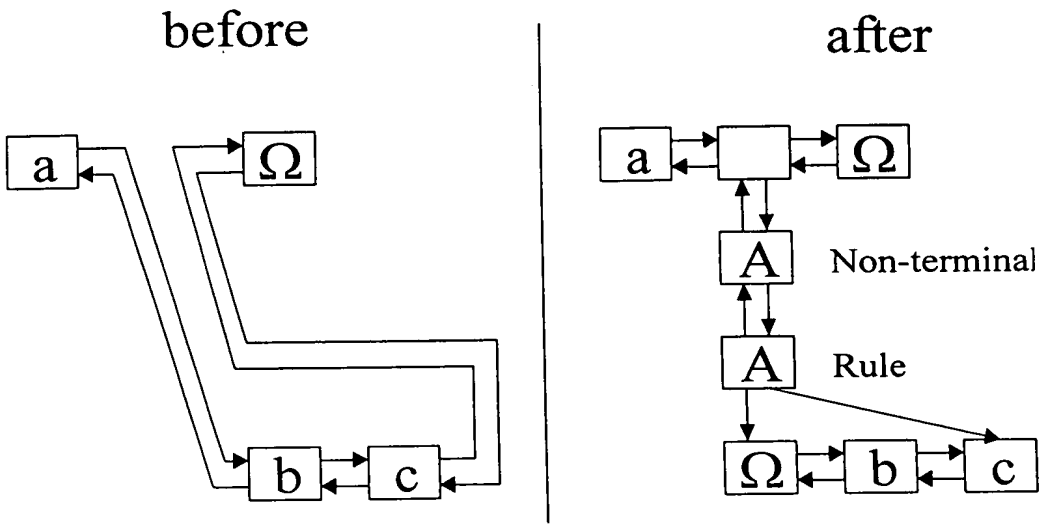


Figure 10

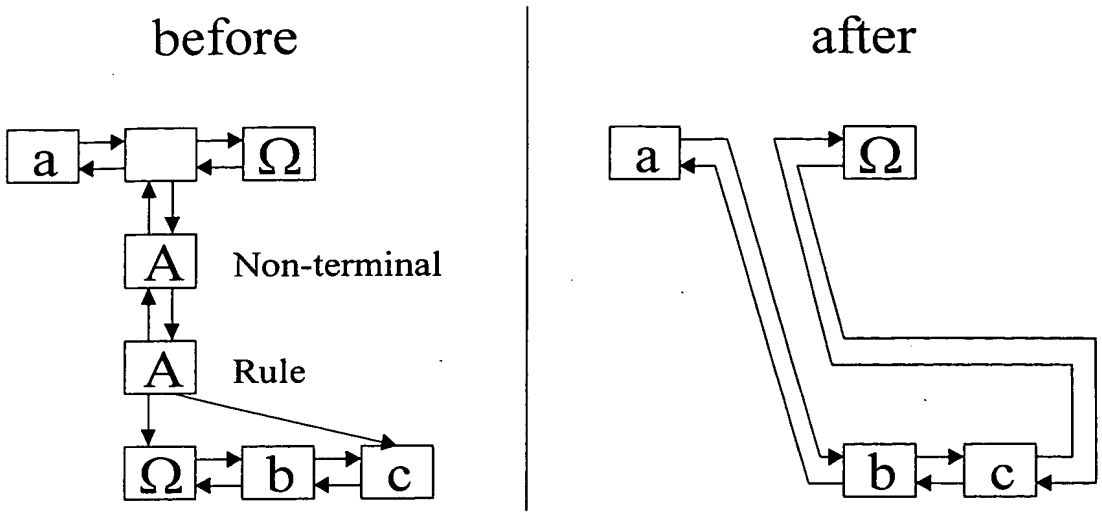


Figure 11

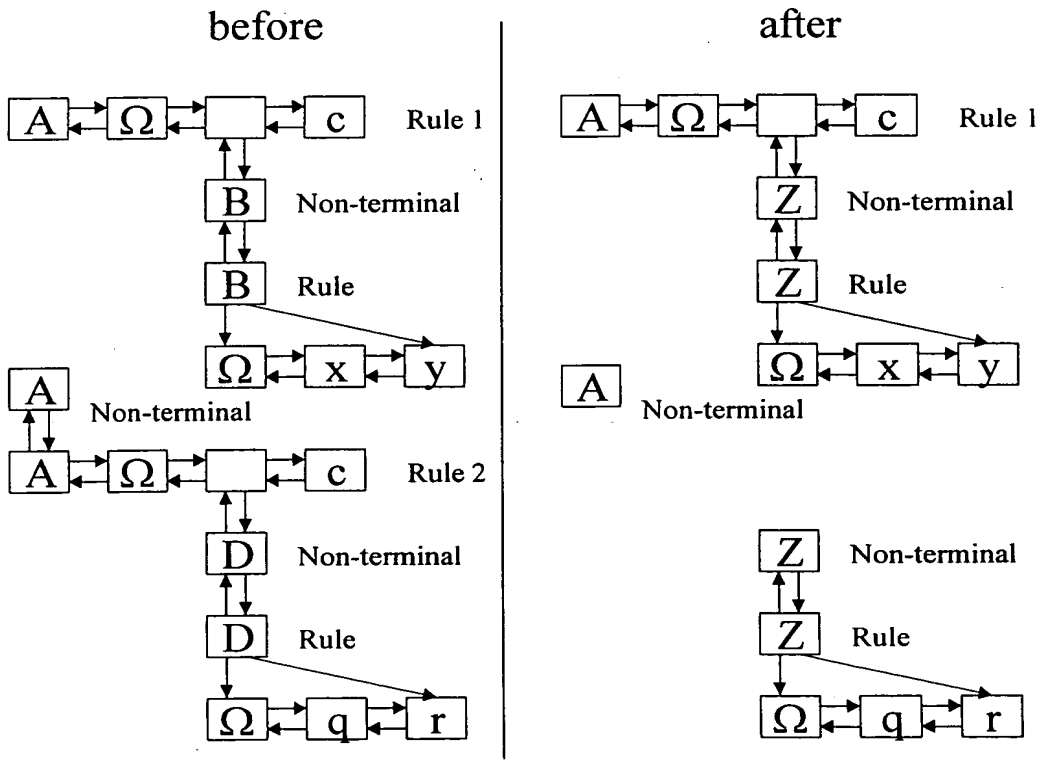


Figure 12

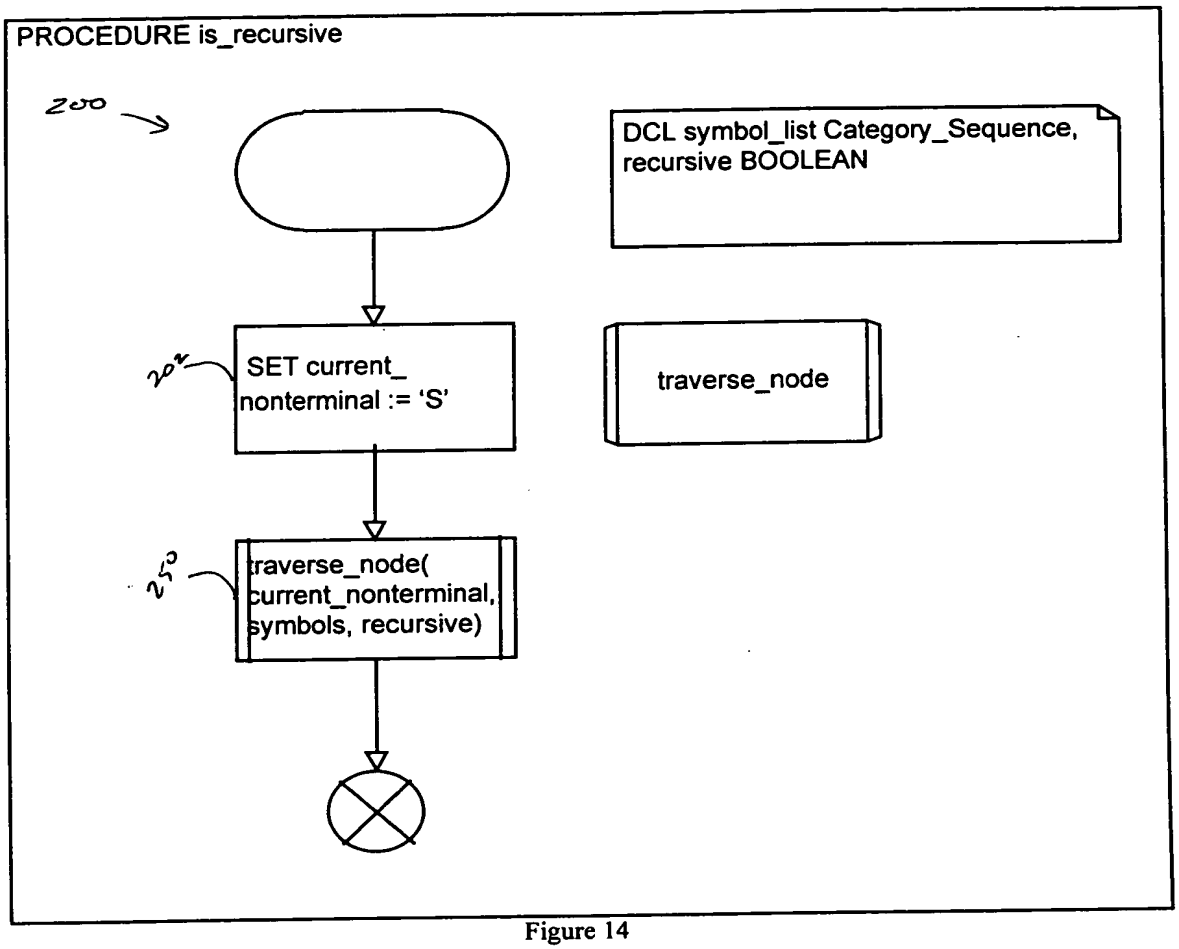

PROCEDURE is_recursive
200
DCL symbol_list Category_Sequence,
recursive BOOLEAN
SET current_
nonterminal := 'S'
traverse_node
traverse_node(
current_nonterminal,
symbols, recursive)

Figure 14

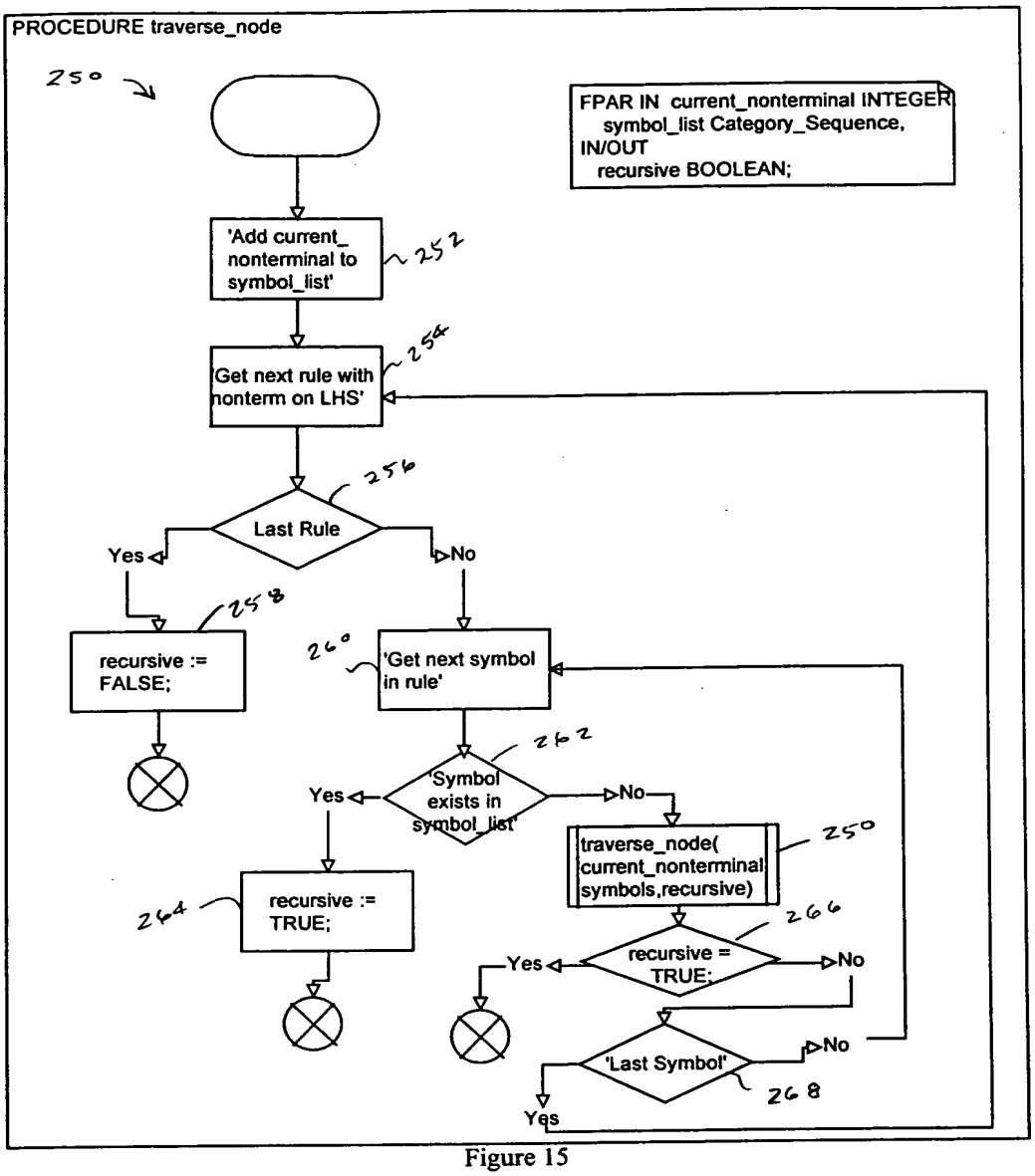

PROCEDURE traverse_node
250
FPAR IN current_nonterminal INTEGER
symbol_list Category_Sequence,
IN/OUT
recursive BOOLEAN;
'Add current_ nonterminal to symbol_list'
252
Get next rule with nonterm on LHS'
254
Last Rule
256
Yes
No
recursive := FALSE;
258
'Get next symbol in rule'
260
'Symbol exists in symbol_list'
262
Yes
No
recursive := TRUE;
264
traverse_node(current_nonterminal symbols,recursive)
250
recursive = TRUE;
266
Yes
No
'Last Symbol'
No
268
Yes

Figure 15